# Formalization of software architectures to study the preservation of properties in composition of components: an example using category theory in the context of Big Data
CAiSE 2021

Annabelle Gillet, Éric Leclercq and Nadine Cullot

March 2021

## 1 Introduction

When architectures evolve and grow, they can combine several smaller parts of architectures developed separately. When building a large scale, complex and distributed architecture, its parts can embed architecture styles on their own. These different cases can result in compositions of smaller architecture parts with their own properties, so formalization should be able to express and control these compositions.

To fill this need, category theory [EM45] is a promising approach: it allows to switch from a model to another or to navigate among abstraction levels, and thus to express various problems from different science fields, such as mathematics, physics or computer science [Spi14, FS19]. By focusing on relations (the morphisms) and compositions, it proposes powerful mechanisms that can be applied to architectures. We focus on studying the conservation or the discarding of properties in compositions of components, by relying on the behaviour of functors combined to preorders. With this formalization, it is possible to deduce:

1. the value for a complex property by knowing values of simple properties that they rely on ;
2. the value of a property for a composition of components by knowing the value taken individually by each component of the composition.

## 2 Essential notions of category theory

This section introduces some basics notions of category theory useful to understand the study of properties in compositions of components: the categories and the functors.

**Definition 2.1.** Category
A **category** $C$ is composed of four fundamentals elements:

1. $\mathrm{Ob}(C)$, a collection of objects ;
2. for each pair $x, y \in \mathrm{Ob}(C)$, a set $\mathrm{Hom}_C(x, y)$ representing **morphisms** from $x$ to $y$, namely a mean to get an object $y$ (the codomain) from an object $x$ (the domain). A morphism $f$ from $x$ to $y$ is noted $f : x \to y$ ;
3. for each $x \in \mathrm{Ob}(C)$, a particular morphism $\mathrm{id}_x$ known as the identity morphism on $x$ ;
4. for each triplet $x, y, z \in \mathrm{Ob}(C)$, a **composition** $\circ : \mathrm{Hom}_C(y, z) \times \mathrm{Hom}_C(x, y) \to \mathrm{Hom}_C(x, z)$. For two morphisms $f : x \to y$ and $g : y \to z$, the composition is noted $g \circ f : x \to z$.

And of two laws:

1. for a morphism $f : x \to y$ with $x, y \in \mathrm{Ob}(C)$, we have $f \circ \mathrm{id}_x = f$ and $\mathrm{id}_y \circ f = f$ ;
2. for $f : w \to x$, $g : x \to y$ and $h : y \to z$ with $w, x, y, z \in \mathrm{Ob}(C)$, we have $(h \circ g) \circ f = h \circ (g \circ f) \in \mathrm{Hom}_C(w, z)$.

In diagrams, categories are represented by boxes. Functors are represented by thick arrows and their effect on objects by dashed arrows. To simplify, identity morphisms are not represented, even if they are always present.

**Definition 2.2.** Functor

A **functor** $F$ is a morphism between two categories. It maps a category $C$ to a category $C'$. It is noted $F : C \to C'$, and acts on:

1. objects: for each object $x \in \mathrm{Ob}(C)$, we have an object $F(x) \in \mathrm{Ob}(C')$ ;
2. morphisms: for each pair $x, y \in \mathrm{Ob}(C)$, we have $F : \mathrm{Hom}_C(x, y) \to \mathrm{Hom}_{C'}(F(x), F(y))$.

A functor must follow two laws to ba valid:

1. the preservation of identities: $\forall x \in \mathrm{Ob}(C)$, $F(\mathrm{id}_x) = \mathrm{id}_{F(x)}$ ;
2. the preservation of composition: for any triplet $x, y, z \in \mathrm{Ob}(C)$ with morphisms $g : x \to y$, $h : y \to z$, we have $F(h \circ g) = F(h) \circ F(g)$.

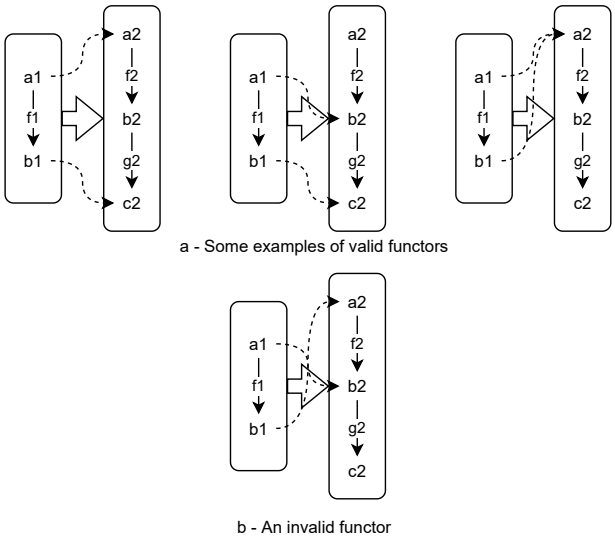


Figure 1: Illustration of functors

The effect of functors on morphisms enforces strong compelings for the validity of the functor. These constraints can be used to deduce several properties. For each object linked to another category with a functor, the functor must also preserve its morphisms. Figure 1 presents some examples of the application of a functor $F$ from a category $C1$ to a category $C2$. In the top subfigure, all the functors are valid: the first transforms objects with $F(a1) = a2$ and $F(b1) = c2$, and the morphism $f1$ is preserved with $F(f1) = g2 \circ f2$. The second is direct and transforms objects with $F(a1) = b2$ and $F(b1) = c2$, and the morphism $f1$ with $F(f1) = g2$. The third relies on the identity morphisms to be valid, and transforms objects with $F(a1) = a2$ and $F(b1) = a2$, and the mrophism $f1$ with $F(f1) = id_{a2}$. The bottom subfigure shows an invalid functor: if this functor transforms objects with $F(a1) = b2$ and $F(b1) = a2$, then there is no mrophism in $C2$ that goes from $b2$ to $a2$ to validate the functor.

# 3 Advanced definitions

Some more advanced concepts of category theory needed to formalize architectures are presented in this section.

**Definition 3.1.** Preordrer

**Preorders** are a special type of category, in which between each pair $x, y \in \text{Ob}(C)$, there exists a unique morphism $f : x \to y$. If there exist $f : x \to y$ and $g : x \to y$, then $f = g$.

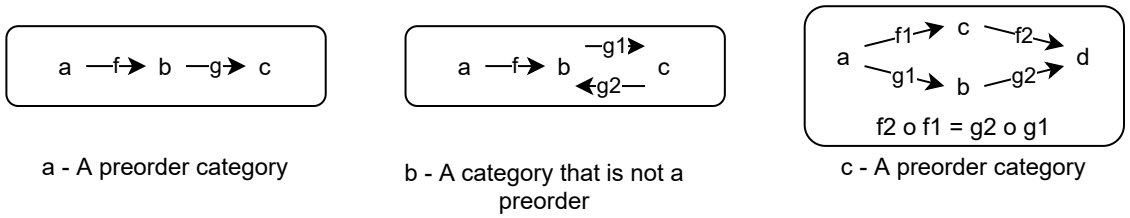


Figure 2: Illustration of preorders

Preorders provide a form of hierarchy in a category. Indeed, for each pair $x, y \in \text{Ob}(C)$, there can be only one morphism, including compositions. The part a of the figure 2 shows a preorder, because there is only one morphism between each pair of objects. The part b is not a preorder, because to get $b$ from $a$, it is possible to use the morphism $f$ or the composition $g_2 \circ g_1 \circ f$. Thus, preorders exclude cycles of morphisms, as it provides many ways to get an object from another object. Nonetheless, it is possible to have multiple morphisms that link a pair of objects in a preorder, at the condition of specifying their equality in an equation, as illustrate in the part c of the figure. In this document, we will omit the equation to simplify figures.

**Definition 3.2.** Power set

The **power sets**, that are sets which contain all the subsets of a given set. In a category, power sets can be organized as preorder, where subsets are objects and morphisms link two subsets if a subset $x$ is integrally included in a subset $y$.

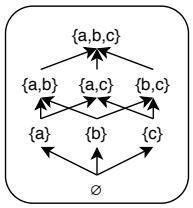


Figure 3: A **power set** for the set $\{a, b, c\}$

An example of power set if given in the figue 3.

**Definition 3.3.** Product of categories

A **product** of two categories is an operation that produces a new cateogry from two existing categories $C1$ and $C2$. The objects of this new category are all the possible pairs $(x, y)$ with $x \in \text{Ob}(C1)$ and $y \in \text{Ob}(C2)$ and the morphisms $(x, y) \to (x', y')$ are pairs $(f, g)$ where $f : x \to x' \in \text{Hom}_{C1}(x, x')$ and $g : y \to y' \in \text{Hom}_{C2}(y, y')$.

Figure 4 shows a product of categories. Part a presents how objects and morphisms are created in the new category. Part b demonstrates how a functor can be applied on the result of a product of categories. Part c shows the diagram simplification of the part b that we will use in the rest of this document to simplify

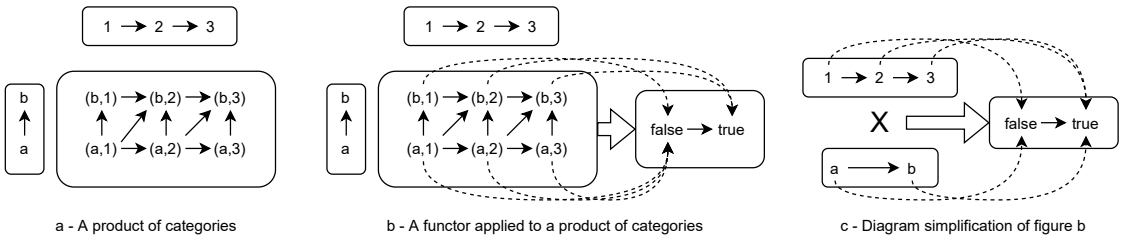


Figure 4: Product of categories and its diagram simplification

diagrams, only when it is enough to represent the effect of the functor. This simplification is valid when a functor is applied on the result of a product of categories between two preorders, and that the category codomain of the functor is also a preorder. It can be used to deduce the effect of the functor on a pair of objects of the product of categories knowing only the targeted object in the new category by each individual object of the pair: the result will be the lowest targeted object in the new category. In our example, if the object $(a, 3)$ would have been transformed in the object *true* in the new category, the diagram simplification could not have been used.

# 4 Principles of the formalization

To formalize an architecture, we use three predefined categories and two predefined functors:

**Definition 4.1.** The category *Components*
In this category, the objects are the components of the architecture, without morphisms to link them, except the identity morphisms.

**Definition 4.2.** The category *Architecture*
In this category, the objects are the components, that are linked by a morphism to represent the interactions. A morphism $f : x \to y$ implies that the component $x$ sends data to the component $y$.

**Definition 4.3.** The category *ComponentsPS*
In this category, the objects are the subsets of the power set of the components.

The category *ComponentsPS* will be used to link components to properties, and to study the conservation in the compositions of components.

**Definition 4.4.** The functor $CA$
The functor $CA$ is defined between the categories *Components* and *Architecture* ($CA : Components \to Architecture$). It links individual components to their position in the architecture. Thus, this functor is used to integrate components in the whole architecture.

**Definition 4.5.** The functor $CCPS$
The functor $CCPS$ is defined between the categories *Components* and *ComponentsPS* ($CCPS : Components \to ComponentsPS$). It links individual components to subsets of the power set with only one element. In this manner, it is possible to study the behaviour of properties in compositions of components.

The figure 5 shows the formalization of a simple architecture with two components $a$ and $b$, represented in the category *Components*. The category *Architecture* is used to define that $a$ sends data to $b$. Each component is sent to its subset in the category *ComponentsPS*. Morphisms of the *ComponentsPS*-category gather components in compositions.

This formalization allows the study of the conservation of properties in compositions of components, by relying on the definition of functors, power sets and products of categories. Functors include in their

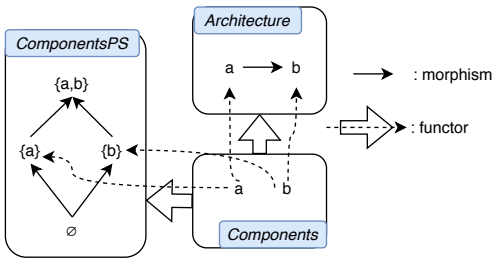


Figure 5: A simple architecture with two components $a$ and $b$

structural definition the notion of composition. Therefore their simple definition, if it is syntactically valid, is sufficient to prove the preservation or loss of a property.

**Definition 4.6.** Formalization of a property

A property is represented by a preorder, in which objects are the different values of the property and morphisms go from the most satisfying value to the least satisfying. The symbol $\top$ is used for component that are not concerned by a property, to avoid impacting compositions of components. $\top$ is the domain of only one morphism, with the most satisfying value as codomain.

Properties can be simple (only with $true$ and $false$ objects), multivalued, or more complex, and resulting of the composition of several other properties: in this case, properties are associated with a product of categories, and this product is linked to the next property with a functor that maps each combination to its signification in the next level property. The complexity of properties can be identified by giving a level to each property: those linked directly to the components of the architecture are level 1, and those depending on product of categories of a higher level. The category $ComponentsPS$ is connected to every property of level one with functors.

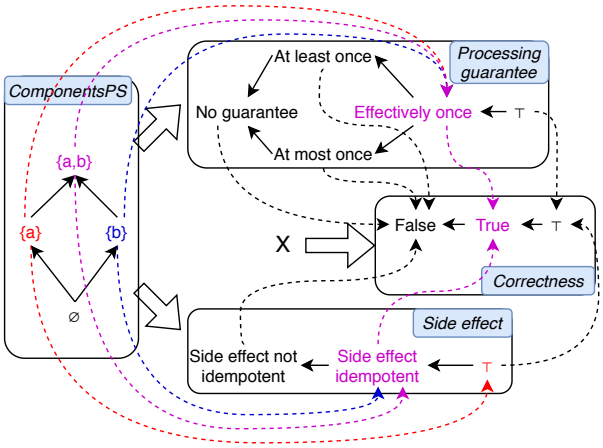


Figure 6: Deduction of a complex property through product of categories

The example depicted in figure 6 studies how the correctness property is supported in an architecture with two components, $a$ and $b$, that use stream processing. The correctness depends on the processing guarantee, as well as the presence or absence of side effects, and if those side effects are idempotent or not. To simplify the schema, the product between the two level one properties is not directly drawn, and is symbolized by a X. Using the power of category theory, we can extract some high-level knowledge from the former representation:

1. by respect to the second property of functors (i.e. $F(h \circ g) = F(h) \circ F(g)$), if a component reaches a lower value for a given property than other components, and if this component is added to the

others, the new set of components will reach the lowest value of those taken by the components for this property. **This is essential in the formalization, as it highlights which composition of components leads to the loss of a property**. In this example, the component $a$ reaches the $\top$ value in the $Side - effect$-category and $b$ reaches the side effect idempotent value, thus, the overall architecture takes the lowest value: the side effect idempotent one ;

2. the product of two level one categories allows to get to a level two category automatically depending on the functor defined between the product and the level two property. In our example, $\{a, b\}$ (the overall architecture), is sent to the value effectively once in the $Processing - guarantee$-category, and to the value side effect idempotent in the $Side - effect$-category. The lowest value reached by the components in the $Correctness$-category is true, thus the correctness property is effective in the architecture.

The definition of functors, and more particularly $F : \mathrm{Hom}_C(x, y) \rightarrow \mathrm{Hom}_{C'}(F(x), F(y))$, allows to deduce the value of a property for a composition of components: **the loss of a property by a component cannot be compensate**. Furthermore, for a complex property, all the functors that map a product of categories to a property with a higher level are defined beforehand. They allow to automatically deduce the value of a complex property for a component or a composition of components, knowing only their value for the level 1 properties.

# 5 Application of the formalization

This formalization allows us to precisely compare the patterns of the Lambda Architecture and the Lambda+ Architecture. To simplify diagrams in this section, the $ComponentsPS$-category has only individual components, as well as possible running compositions of the architecture, and only the links from components to properties that do not lead to $\top$ are represented.

## 5.1 The Lambda Architecture

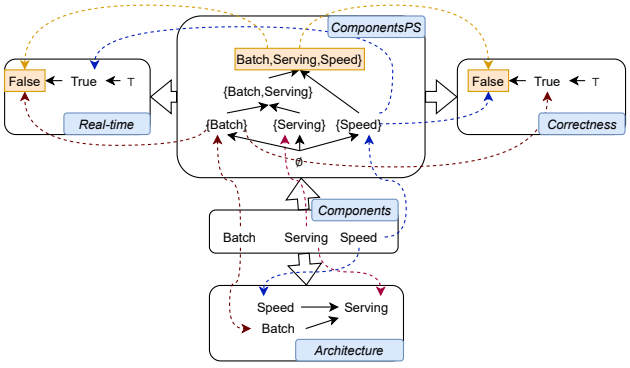


Figure 7: Formalization of the Lambda Architecture pattern

The figure 7 shows the formalization of the Lambda Architecture pattern. The serving layer is neutral concerning the correctness and the real-time properties, as it only puts data that have been computed in the other layers to disposal. The speed layer has the real-time property, but not the correctness, and the batch layer is the opposite, with the correctness property but not the real-time. By supporting the real-time property only in the speed layer, and the correctness property only in the batch layer, the running composition of components does not support these properties simultaneously. It results in an architecture that produces incorrect results continuously, and corrects it periodically in a batch fashion.

Using the category theory, we can extract some high-level knowledge from the known facts, and formally prove the loss of properties in the overall architecture. The morphisms inside each category are defined as follows (with the notation $B$ = Batch, $Se$ = Serving and $Sp$ = Speed, morphisms with $\emptyset$ as domain and identity morphisms in the $ComponentsPS$-category are omitted):

| $CPS$ ($ComponentsPS$) | $C$ ($Correctness$) | $RT$ ($Real-time$) |
|---|---|---|
| $B - BSe : B \to BSe$ | $\top - t : \top \to True$ | $\top - t : \top \to True$ |
| $Se - BSe : Se \to BSe$ | $t - f : True \to False$ | $t - f : True \to False$ |
| $BSe - BSeSp : BSe \to BSeSp$ | $id_\top : \top \to \top$ | $id_\top : \top \to \top$ |
| $Sp - BSeSp : Sp \to BSeSp$ | $id_t : True \to True$ | $id_t : True \to True$ |
| | $id_f : False \to False$ | $id_f : False \to False$ |

And the effect on the objects of functors that link the $ComponentsPS$-category to the categories of the properties:

| $CPS - C$ ($Correctness$) | $CPS - RT$ ($Real - time$) |
|---|---|
| $B \to True$ | $B \to False$ |
| $Se \to \top$ | $Se \to \top$ |
| $Sp \to False$ | $Sp \to True$ |

### 5.1.1 Correctness property

From these given facts, we want to deduce the value taken by the first composition of components $\{Batch, Serving\}$ for the correctness property. For this, we have to resolve the effect of the functor $CPS-C$ on the morphisms:

$$CPS - C : \underline{\text{Hom}_{CPS}(B, BSe)} \to \text{Hom}_C(\underline{CPS - C(B)}, CPS - C(BSe))$$
$$CPS - C : \underline{\text{Hom}_{CPS}(Se, BSe)} \to \text{Hom}_C(\underline{CPS - C(Se)}, CPS - C(BSe))$$

where the underlined elements are known. To establish the value of $CPS - C(BSe)$, we have to find two morphisms in the category $C$ that would have the same codomain: one with the domain $True$ ($CPS-C(B)$), the other with the domain $\top$ ($CPS - C(Se)$). Only the pair ($\top - t$, $id_t$) satisfies the requirements. As the codomain is $True$, it allows us to deduce that $CPS - C : BSe \to True$. Thus, the composition $\{Batch, Serving\}$ yields $True$ for the correctness property.

We use the same mechanism to deduce the value taken by the overall architecture for the same property:

$$CPS - C : \underline{\text{Hom}_{CPS}(Sp, BSeSp)} \to \text{Hom}_C(\underline{CPS - C(Sp)}, CPS - C(BSeSp))$$
$$CPS - C : \underline{\text{Hom}_{CPS}(BSe, BSeSp)} \to \text{Hom}_C(\underline{CPS - C(BSe)}, CPS - C(BSeSp))$$

This time, the pair of morphisms that meets the requirements is ($t - f$, $id_f$). As the common codomain is $False$, it allows us to deduce that the overall architecture yields $False$ for the correctness property.

### 5.1.2 Real-time property

We apply the same reasoning to deduce the value taken by the composition $\{Batch, Serving\}$ for the real-time property. For this, we have to resolve the effect of the functor $CPS - RT$ on the morphisms:

$$CPS - RT : \underline{\text{Hom}_{CPS}(B, BSe)} \to \text{Hom}_{RT}(\underline{CPS - RT(B)}, CPS - RT(BSe))$$
$$CPS - RT : \underline{\text{Hom}_{CPS}(Se, BSe)} \to \text{Hom}_{RT}(\underline{CPS - RT(Se)}, CPS - RT(BSe))$$

The pair of morphisms that meets the requirements is ($t - f$, $id_f$). As the common codomain is $False$, it allows us to deduce that the the composition $\{Batch, Serving\}$ yields $False$ for the real-time property.

We use the same mechanism to deduce the value taken by the overall architecture for the same property:

$$CPS - RT : \underline{\text{Hom}_{CPS}(Sp, BSeSp)} \to \text{Hom}_{RT}(\underline{CPS - RT(Sp)}, CPS - RT(BSeSp))$$
$$CPS - RT : \underline{\text{Hom}_{CPS}(BSe, BSeSp)} \to \text{Hom}_{RT}(\underline{CPS - RT(BSe)}, CPS - RT(BSeSp))$$

The pair of morphisms that meets the requirements is ($t - f$, $id_f$). As the common codomain is $False$, it allows us to deduce that the overall architecture yields also $False$ for the real-time property.

## 5.2 The Lambda+ Architecture

The figure 8 shows the application of the formalization on the Lambda+ Architecture pattern. Two running compositions of components are possible: with or without the master dataset, as it is only linked to the streaming ETL when data have to be reprocessed. As the Lambda+ integrates the advances of stream processing systems, there is no component that invalidates the correctness property, as long as the side effects are idempotent (if they are present), and that the effectively-once property is available. So contrary to the Lambda Architecture, this property holds for the overall architecture. Concerning the real-time property, only the master dataset induces its loss, but in exchange the Lambda+ activates the fault-tolerance property. It is an acceptable trade-off, as the master dataset is only used in emergency cases, to reprocess data after a failure or when the needs change.

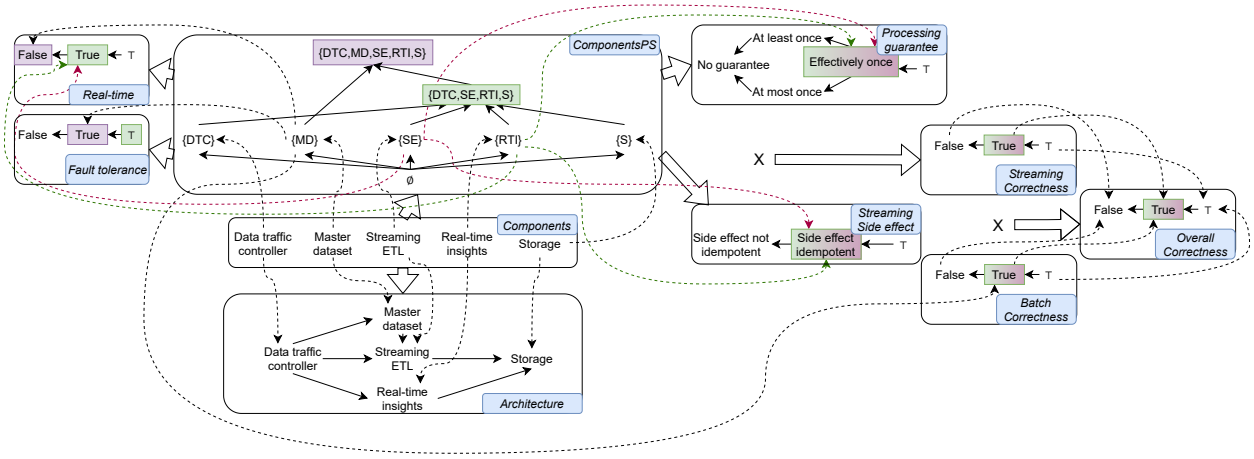


Figure 8: Formalization of the Lambda+ Architecture pattern

The morphisms inside each category are defined as follows (with the notation $Dtc$ = Data traffic controller, $Se$ = Streaming ETL, $Rti$ = Real-time insights, $Md$ = Master dataset and $S$ = Storage, morphisms with $\emptyset$ as domain and identity morphisms in the $ComponentsPS$-category are omitted, as well as those that lead to a composition that cannot be obtained in the architecture):

$$
\begin{array}{r|l}
CPS \\ (ComponentsPS)
& \begin{array}{l}
Dtc - Md : Dtc \to DtcMd \\
Dtc - Se : Dtc \to DtcSe \\
Dtc - Rti : Dtc \to DtcRti \\
DtcMd - Se : DtcMd \to DtcMdSe \\
DtcSe - S : DtcSe \to DtcSeS \\
DtcRti - S : DtcRti \to DtcRtiS \\
DtcRtiS - Se : DtcRtiS \to DtcRtiSeS \\
DtcSeS - Rti : DtcSeS \to DtcRtiSeS \\
DtcRtiSeS - Md : DtcRtiSeS \to DtcRtiSeSMd
\end{array}
\end{array}
\qquad
\begin{array}{r|l}
PG \\ (Processing- \\ guarantee)
& \begin{array}{l}
\top - e : \top \to effectively - once \\
e - l : effectively - once \to at - least - once \\
e - m : effectively - once \to at - most - once \\
l - n : at - least - once \to no - guarantee \\
m - n : at - most - once \to no - guarantee \\
id_\top : \top \to \top \\
id_e : effectively - once \to effectively - once \\
id_l : at - least - once \to at - least - once \\
id_m : at - most - once \to at - most - once \\
id_n : no - guarantee \to no - guarantee
\end{array}
\end{array}
$$

$$
\begin{array}{r|l}
FT \\ (Fault- \\ tolerance)
& \begin{array}{l}
\top - t : \top \to True \\
t - f : True \to False \\
id_\top : \top \to \top \\
id_t : True \to True \\
id_f : False \to False
\end{array}
\end{array}
\qquad
\begin{array}{r|l}
RT \\ (Real- \\ time)
& \begin{array}{l}
\top - t : \top \to True \\
t - f : True \to False \\
id_\top : \top \to \top \\
id_t : True \to True \\
id_f : False \to False
\end{array}
\end{array}
\qquad
\begin{array}{r|l}
SE \\ (Side- \\ effect)
& \begin{array}{l}
\top - i : \top \to idempotent \\
i - ni : idempotent \to not - idempotent \\
id_\top : \top \to \top \\
id_i : idempotent \to idempotent \\
id_{ni} : not - idempotent \to not - idempotent
\end{array}
\end{array}
$$

$$
\begin{array}{r|l}
SC \\ (Streaming- \\ correctness)
& \begin{array}{l}
\top - t : \top \to True \\
t - f : True \to False \\
id_\top : \top \to \top \\
id_t : True \to True \\
id_f : False \to False
\end{array}
\end{array}
\qquad
\begin{array}{r|l}
BC \\ (Batch- \\ correctness)
& \begin{array}{l}
\top - t : \top \to True \\
t - f : True \to False \\
id_\top : \top \to \top \\
id_t : True \to True \\
id_f : False \to False
\end{array}
\end{array}
\qquad
\begin{array}{r|l}
OC \\ (Overall- \\ correctness)
& \begin{array}{l}
\top - t : \top \to True \\
t - f : True \to False \\
id_\top : \top \to \top \\
id_t : True \to True \\
id_f : False \to False
\end{array}
\end{array}
$$

The effect of the functors from the $ComponentsPS$-category to the categories of properties of level one are defined below:

$$
\begin{array}{r|l}
CPS - RT \\ (Real - time)
& \begin{array}{l}
Dtc \to True \\
Md \to False \\
Se \to True \\
Rti \to True \\
S \to \top
\end{array}
\end{array}
\qquad
\begin{array}{r|l}
CPS - FT \\ (Fault- \\ tolerance)
& \begin{array}{l}
Dtc \to \top \\
Md \to True \\
Se \to \top \\
Rti \to \top \\
S \to \top
\end{array}
\end{array}
\qquad
\begin{array}{r|l}
CPS - PG \\ (Processing- \\ guarantee)
& \begin{array}{l}
Dtc \to effectively - once \\
Md \to \top \\
Se \to effectively - once \\
Rti \to effectively - once \\
S \to \top
\end{array}
\end{array}
$$

$$\begin{array}{r|l} CPS-SE \\ (Side- \\ effect) \end{array} \left| \begin{array}{l} Dtc \to idempotent \\ Md \to \top \\ Se \to idempotent \\ Rti \to idempotent \\ S \to \top \end{array} \right. \qquad \begin{array}{r|l} CPS-PG \\ (Batch- \\ correctness) \end{array} \left| \begin{array}{l} Dtc \to \top \\ Md \to True \\ Se \to \top \\ Rti \to \top \\ S \to \top \end{array} \right.$$

We also define the effect on objects of functors applied on products of categories:

$$PG \times SE - SC \left| \begin{array}{l} (\top, \top) \to \top \\ (\top, i) \to \top \\ (\top, ni) \to \top \\ (e, \top) \to True \\ (e, i) \to True \\ (e, ni) \to False \\ (l, i) \to False \\ (m, i) \to False \\ (l, ni) \to False \\ (m, ni) \to False \\ (n, i) \to False \\ (n, ni) \to False \end{array} \right. \qquad SC \times BC - OC \left| \begin{array}{l} (\top, \top) \to \top \\ (\top, True) \to True \\ (True, \top) \to True \\ (True, True) \to True \\ (True, False) \to False \\ (False, True) \to False \\ (False, False) \to False \end{array} \right.$$

Now that all the known facts are defined, we can use the same mechanism as the one demonstrated on the Lambda Architecture to deduce the properties of the Lambda+ Architecture. To avoid having to deduce the value of a property pair of morphisms by pair of morphisms, we use a shortcut to deduce it directly from a set of morphisms, that go from each individual components of the composition to the composition itself. As the $ComponentsPS$-category is a power-set and a preorder, it means that as long as the component is in the composition of components, their is a composition of morphisms that leads to it (see definition 3.2).

For example, the morphism from the component $Dtc$ to the composition $DtcRtiSeS$ exists and can be obtained with :

$$Hom_{CPS}(Dtc, DtcRtiSeS) = Hom_{CPS}(DtcSeS - Rti) \circ Hom_{CPS}(DtcSe - S) \circ Hom_{CPS}(Dtc - Se)$$

.

### 5.2.1 Fault-tolerance property

We start by deducing the value taken by the composition {*Data traffic controller, Streaming ETL, Real-time insights, Storage*} for the fault-tolerance property. We have to resolve the effect of the functor $CPS - FT$ on the morphisms:

$$\begin{array}{r} CPS-FT : \underline{\mathrm{Hom}_{CPS}(Dtc, DtcRtiSeS)} \to \mathrm{Hom}_{FT}(\underline{CPS-FT(Dtc)}, CPS-FT(DtcRtiSeS)) \\ CPS-FT : \underline{\mathrm{Hom}_{CPS}(Rti, DtcRtiSeS)} \to \mathrm{Hom}_{FT}(\underline{CPS-FT(Rti)}, CPS-FT(DtcRtiSeS)) \\ CPS-FT : \underline{\mathrm{Hom}_{CPS}(Se, DtcRtiSeS)} \to \mathrm{Hom}_{FT}(\underline{CPS-FT(Se)}, CPS-FT(DtcRtiSeS)) \\ CPS-FT : \underline{\mathrm{Hom}_{CPS}(S, DtcRtiSeS)} \to \mathrm{Hom}_{FT}(\underline{CPS-FT(S)}, CPS-FT(DtcRtiSeS)) \end{array}$$

In this case, all the morphisms have $\top$ as domain: it means that the components cannot activate the property by themselves, and that they will not invalidate the property if they are composed with components that support the property. We have three morphisms valid for the fault-tolerance property: $id_\top$, $\top - t$ and $t - f \circ \top - t$. We keep the more advantageous ($id_\top$), and define that: $CPS - FT : DtcRtiSeS \to \top$.

We use the same mechanism to deduce the value taken by composition {*Data traffic controller, Streaming ETL, Real-time insights, Storage, Master dataset*} for the same property:

$$\begin{array}{r} CPS-FT : \underline{\mathrm{Hom}_{CPS}(DtcRtiSeS, DtcRtiSeSMd)} \to \mathrm{Hom}_{FT}(\underline{CPS-FT(DtcRtiSeS)}, CPS-FT(DtcRtiSeSMd)) \\ CPS-FT : \underline{\mathrm{Hom}_{CPS}(Md, DtcRtiSeSMd)} \to \mathrm{Hom}_{FT}(\underline{CPS-RT(Md)}, CPS-FT(DtcRtiSeSMd)) \end{array}$$

The pair of morphisms that meets the requirements is ($\top - t$, $id_t$). As the common codomain is $True$, it allows us to deduce that the composition yields $True$ for the fault-tolerance property, so $CPS - FT : DtcRtiSeSMd \to True$. Thus, the Lambda+ can activate the fault-tolerance property when the master dataset is linked to the streaming ETL component.

### 5.2.2 Real-time property

We study the value taken by the composition {*Data traffic controller, Streaming ETL, Real-time insights, Storage*} for the real-time property. We have to resolve the effect of the functor $CPS-RT$ on the morphisms:

$$\begin{aligned}
CPS-RT : \underline{\mathrm{Hom}_{CPS}(Dtc, DtcRtiSeS)} &\to \mathrm{Hom}_{RT}(\underline{CPS-RT(Dtc)}, CPS-RT(DtcRtiSeS)) \\
CPS-RT : \underline{\mathrm{Hom}_{CPS}(Rti, DtcRtiSeS)} &\to \mathrm{Hom}_{RT}(\underline{CPS-RT(Rti)}, CPS-RT(DtcRtiSeS)) \\
CPS-RT : \underline{\mathrm{Hom}_{CPS}(Se, DtcRtiSeS)} &\to \mathrm{Hom}_{RT}(\underline{CPS-RT(Se)}, CPS-RT(DtcRtiSeS)) \\
CPS-RT : \underline{\mathrm{Hom}_{CPS}(S, DtcRtiSeS)} &\to \mathrm{Hom}_{RT}(\underline{CPS-RT(S)}, CPS-RT(DtcRtiSeS))
\end{aligned}$$

The pair of morphisms that meets the requirements is $(\top - t,\ id_t)$. As the common codomain is $True$, it allows us to deduce that the composition yields $True$ for the real-time property, so $CPS-RT : DtcRtiSeS \to True$.

We use the same mechanism to deduce the value taken by composition {*Data traffic controller, Streaming ETL, Real-time insights, Storage, Master dataset*} for the same property:

$$\begin{aligned}
CPS-RT : \underline{\mathrm{Hom}_{CPS}(DtcRtiSeS, DtcRtiSeSMd)} &\to \mathrm{Hom}_{RT}(\underline{CPS-RT(DtcRtiSeS)}, CPS-RT(DtcRtiSeSMd)) \\
CPS-RT : \underline{\mathrm{Hom}_{CPS}(Md, DtcRtiSeSMd)} &\to \mathrm{Hom}_{RT}(\underline{CPS-RT(Md)}, CPS-RT(DtcRtiSeSMd))
\end{aligned}$$

The pair of morphisms that meets the requirements is $(t - f,\ id_f)$. As the common codomain is $False$, it allows us to deduce that the composition yields $False$ for the real-time property, so $CPS-RT : DtcRtiSeSMd \to False$. Indeed, it is the required trade-off to activate the fault-tolerance property: by merging the master dataset in the composition, we loose the real-time capability to perform a occasional reprocessing, if a failure happen or if the needs evolve.

### 5.2.3 Processing-guarantee property

We study the value taken by the composition {*Data traffic controller, Streaming ETL, Real-time insights, Storage*} for the processing-guarantee property. We have to resolve the effect of the functor $CPS-PG$ on the morphisms:

$$\begin{aligned}
CPS-PG : \underline{\mathrm{Hom}_{CPS}(Dtc, DtcRtiSeS)} &\to \mathrm{Hom}_{PG}(\underline{CPS-PG(Dtc)}, CPS-PG(DtcRtiSeS)) \\
CPS-PG : \underline{\mathrm{Hom}_{CPS}(Rti, DtcRtiSeS)} &\to \mathrm{Hom}_{PG}(\underline{CPS-PG(Rti)}, CPS-PG(DtcRtiSeS)) \\
CPS-PG : \underline{\mathrm{Hom}_{CPS}(Se, DtcRtiSeS)} &\to \mathrm{Hom}_{PG}(\underline{CPS-PG(Se)}, CPS-PG(DtcRtiSeS)) \\
CPS-PG : \underline{\mathrm{Hom}_{CPS}(S, DtcRtiSeS)} &\to \mathrm{Hom}_{PG}(\underline{CPS-PG(S)}, CPS-PG(DtcRtiSeS))
\end{aligned}$$

The pair of morphisms that meets the requirements is $(\top - e,\ id_e)$. As the common codomain is $effectively-once$, it allows us to deduce that the composition support the $effectively-once$ guarantee in components using the stream processing, so $CPS-PG : DtcRtiSeS \to effectively-once$.

We use the same mechanism to deduce the value taken by composition {*Data traffic controller, Streaming ETL, Real-time insights, Storage, Master dataset*} for the same property:

$$\begin{aligned}
CPS-PG : \underline{\mathrm{Hom}_{CPS}(DtcRtiSeS, DtcRtiSeSMd)} &\to \mathrm{Hom}_{PG}(\underline{CPS-PG(DtcRtiSeS)}, CPS-PG(DtcRtiSeSMd)) \\
CPS-PG : \underline{\mathrm{Hom}_{CPS}(Md, DtcRtiSeSMd)} &\to \mathrm{Hom}_{PG}(\underline{CPS-PG(Md)}, CPS-PG(DtcRtiSeSMd))
\end{aligned}$$

The pair of morphisms that meets the requirements is $(\top - e,\ id_e)$. As the common codomain is $effectively-once$, this composition support also the $effectively-once$ guarantee in components using the stream processing, so $CPS-PG : DtcRtiSeSMd \to effectively-once$.

### 5.2.4 Side-effect property

We study the value taken by the composition {*Data traffic controller, Streaming ETL, Real-time insights, Storage*} for the side-effect property. We have to resolve the effect of the functor $CPS-SE$ on the morphisms:

$$\begin{aligned}
CPS-SE : \underline{\mathrm{Hom}_{CPS}(Dtc, DtcRtiSeS)} &\to \mathrm{Hom}_{SE}(\underline{CPS-SE(Dtc)}, CPS-SE(DtcRtiSeS)) \\
CPS-SE : \underline{\mathrm{Hom}_{CPS}(Rti, DtcRtiSeS)} &\to \mathrm{Hom}_{SE}(\underline{CPS-PG(Rti)}, CPS-SE(DtcRtiSeS)) \\
CPS-SE : \underline{\mathrm{Hom}_{CPS}(Se, DtcRtiSeS)} &\to \mathrm{Hom}_{SE}(\underline{CPS-SE(Se)}, CPS-SE(DtcRtiSeS)) \\
CPS-SE : \underline{\mathrm{Hom}_{CPS}(S, DtcRtiSeS)} &\to \mathrm{Hom}_{SE}(\underline{CPS-SE(S)}, CPS-SE(DtcRtiSeS))
\end{aligned}$$

The pair of morphisms that meets the requirements is $(\top - i,\ id_i)$. As the common codomain is *idempotent*. It allows us to deduce that the composition can have *idempotent* side-effects in components using stream processing systems, so $CPS - SE : DtcRtiSeS \rightarrow idempotent$.

We use the same mechanism to deduce the value taken by composition {*Data traffic controller, Streaming ETL, Real-time insights, Storage, Master dataset*} for the same property:

$$\begin{array}{r} CPS - SE : \underline{\mathrm{Hom}_{CPS}(DtcRtiSeS, DtcRtiSeSMd)} \rightarrow \mathrm{Hom}_{SE}(\underline{CPS - SE(DtcRtiSeS)}, CPS - SE(DtcRtiSeSMd)) \\ CPS - SE : \underline{\mathrm{Hom}_{CPS}(Md, DtcRtiSeSMd)} \rightarrow \mathrm{Hom}_{SE}(\underline{CPS - SE(Md)}, CPS - SE(DtcRtiSeSMd)) \end{array}$$

The pair of morphisms that meets the requirements is $(\top - i,\ id_i)$. As the common codomain is *idempotent*, this composition can also have *idempotent* side-effects in components using stream processing systems, $CPS - SE : DtcRtiSeSMd \rightarrow idempotent$.

### 5.2.5 Batch-correctness property

We study the value taken by the composition {*Data traffic controller, Streaming ETL, Real-time insights, Storage*} for the batch-correctness property. We have to resolve the effect of the functor $CPS - BC$ on the morphisms:

$$\begin{array}{r} CPS - BC : \underline{\mathrm{Hom}_{CPS}(Dtc, DtcRtiSeS)} \rightarrow \mathrm{Hom}_{BC}(\underline{CPS - BC(Dtc)}, CPS - BC(DtcRtiSeS)) \\ CPS - BC : \underline{\mathrm{Hom}_{CPS}(Rti, DtcRtiSeS)} \rightarrow \mathrm{Hom}_{BC}(\underline{CPS - BC(Rti)}, CPS - BC(DtcRtiSeS)) \\ CPS - BC : \underline{\mathrm{Hom}_{CPS}(Se, DtcRtiSeS)} \rightarrow \mathrm{Hom}_{BC}(\underline{CPS - BC(Se)}, CPS - BC(DtcRtiSeS)) \\ CPS - BC : \underline{\mathrm{Hom}_{CPS}(S, DtcRtiSeS)} \rightarrow \mathrm{Hom}_{BC}(\underline{CPS - BC(S)}, CPS - BC(DtcRtiSeS)) \end{array}$$

In this case, all the morphisms have $\top$ as domain: these components are stream processing-oriented, so they are not concerned by the batch correctness. We have three morphisms valid for the fault-tolerance property: $id_\top, \top - t$ and $t - f \circ \top - t$. We keep the more advantageous $(id_\top)$, and define that: $CPS - BC : DtcRtiSeS \rightarrow \top$.

We use the same mechanism to deduce the value taken by composition {*Data traffic controller, Streaming ETL, Real-time insights, Storage, Master dataset*} for the same property:

$$\begin{array}{r} CPS - BC : \underline{\mathrm{Hom}_{CPS}(DtcRtiSeS, DtcRtiSeSMd)} \rightarrow \mathrm{Hom}_{BC}(\underline{CPS - BC(DtcRtiSeS)}, CPS - BC(DtcRtiSeSMd)) \\ CPS - BC : \underline{\mathrm{Hom}_{CPS}(Md, DtcRtiSeSMd)} \rightarrow \mathrm{Hom}_{BC}(\underline{CPS - BC(Md)}, CPS - BC(DtcRtiSeSMd)) \end{array}$$

The pair of morphisms that meets the requirements is $(\top - t,\ id_t)$. As the common codomain is $True$, the batch-correctness is valid in the composition, so $CPS - BC : DtcRtiSeSMd \rightarrow True$.

### 5.2.6 Streaming-correctness property

The streaming-correctness property is a level two property, we first need to deduce the two level one properties on which it depends: the processing-guarantee and the side-effect properties. For the composition {*Data traffic controller, Streaming ETL, Real-time insights, Storage*}, we found:

$$\begin{array}{c} CPS - SE : DtcRtiSeS \rightarrow idempotent \\ CPS - PG : DtcRtiSeS \rightarrow effectively - once \end{array}$$

To find the value in the streaming-correctness property, we use the effect of the functor on the pair $(effectively - once, idempotent)$. We find $PG \times SE - SC : (e, i) \rightarrow True$. Thus, the streaming-correctness property holds for this composition.

We rely on the same mechanism to deduce the value of the streaming-correctness property for the composition {*Data traffic controller, Streaming ETL, Real-time insights, Storage, Master dataset*}. We have:

$$\begin{array}{c} CPS - SE : DtcRtiSeSMd \rightarrow idempotent \\ CPS - PG : DtcRtiSeSMd \rightarrow effectively - once \end{array}$$

We find $PG \times SE - SC : (e, i) \rightarrow True$. The streaming-correctness property holds also for this composition.

#### 5.2.7 Overall-correctness property

The final property to study is the overall-correctness, that is valid only if the streaming and the batch correctness are valid. For the composition {*Data traffic controller, Streaming ETL, Real-time insights, Storage*}, we found:

$$\begin{gathered} CPS - SE : DtcRtiSeS \to idempotent \\ CPS - PG : DtcRtiSeS \to effectively - once \\ PG \times SE - SC : (e, i) \to True \\ CPS - BC : DtcRtiSeS \to \top \end{gathered}$$

We look for the effect on the pair $(True, \top)$ for the functor $SC \times BC - OC$. We find $SC \times BC - OC : (True, \top) \to True$, so the composition respect the overall-correctness property.

We rely on the same mechanism to deduce the value of this property for the composition {*Data traffic controller, Streaming ETL, Real-time insights, Storage, Master dataset*}. We have:

$$\begin{gathered} CPS - SE : DtcRtiSeSMd \to idempotent \\ CPS - PG : DtcRtiSeSMd \to effectively - once \\ PG \times SE - SC : (e, i) \to True \\ CPS - BC : DtcRtiSeSMd \to True \end{gathered}$$

We find $SC \times BC - OC : (True, True) \to True$. The overall-correctness property cannot be lost in this pattern, contrary to the Lambda Architecture.

## References


[EM45] Samuel Eilenberg and Saunders MacLane. General theory of natural equivalences. *Transactions of the American Mathematical Society*, 58(2):231–294, 1945.

[FS19] Brendan Fong and David I Spivak. *An invitation to applied category theory: seven sketches in compositionality*. Cambridge University Press, 2019.

[Spi14] David I Spivak. *Category theory for the sciences*. MIT Press, 2014.